A Categorical Semantics for Linear Logical Frameworks

Matthijs Vákár

Department of Computer Science,
University of Oxford, Oxford, United Kingdom

Abstract. A type theory is presented that combines (intuitionistic) linear types with type dependency, thus properly generalising both intuitionistic dependent type theory and full linear logic. A syntax and complete categorical semantics are developed, the latter in terms of (strict) indexed symmetric monoidal categories with comprehension. Various optional type formers are treated in a modular way. In particular, we will see that the historically much-debated multiplicative quantifiers and identity types arise naturally from categorical considerations. These new multiplicative connectives are further characterised by several identities relating them to the usual connectives from dependent type theory and linear logic. Finally, one important class of models, given by families with values in some symmetric monoidal category, is investigated in detail.

1 Introduction

Starting from Church's simply typed λ-calculus (or intuitionistic propositional type theory), two extensions in perpendicular directions depart:

- following the Curry-Howard propositions-as-types interpretation *dependent type theory* (DTT) [1] extends the simply typed λ-calculus from a proof-calculus of intuitionistic propositional logic to one for predicate logic;
- *linear logic* [2] gives a more detailed resource sensitive analysis, exposing precisely how many times each assumption is used in proofs.

A combined *linear dependent type theory* is one of the interesting directions to explore to gain a more fine-grained understanding of *homotopy type theory* [3] from a computer science point of view, explaining its flow of information. Indeed, many of the usual settings for computational semantics are naturally linear in character, either because they arise as !-co-Kleisli categories (coherence space and game semantics) or for more fundamental reasons (quantum computation).

Combining dependent types and linear types is a non-trivial task, however, and despite some work by various authors that we shall discuss, the precise relationship between the two systems remains poorly understood. The discrepancy between linear and dependent types is the following.

- The lack of structural rules in *linear type theory* forces us to refer to each variable precisely once - for a sequent $x : A \vdash t : B$, x occurs uniquely in t.
- In *dependent type theory*, types can have free variables - $x : A \vdash B$ type, where x is free in B. Crucially, if $x : A \vdash t : B$, x may also be free in t.

What does it mean for x to occur uniquely in t in a dependent setting? Do we count its occurrence in B? The usual way out, which we shall follow too, is to restrict type dependency on intuitionistic terms. Although this seems very limiting - for instance, we do not obtain an equivalent of the Girard translation, embedding DTT in the resulting system -, it is not clear that there is a reasonable alternative. Moreover, as even this limited scenario has not been studied extensively, we hope that a semantic analysis, which was so far missing entirely, may shed new light on the old mystery of linear type dependency.

Historically, Girard's early work in linear logic already makes movements to extend a linear analysis to predicate logic. Although it talks about first-order quantifiers, the analysis appears to have stayed rather superficial, omitting the identity predicates which, in a way, are what make first-order logic tick. Closely related is that an account of internal quantification, or a linear variant of Martin-Löf's type theory, was missing, let alone a Curry-Howard correspondence.

Later, linear types and dependent types were first combined in a Linear Logical Framework [4], where a syntax was presented that extends a Logical Framework with linear types (that depend on terms of intuitionistic types). This has given rise to a line of work in the computer science community [5,6,7]. All the work seems to be syntactic in nature, however, and seems to be mostly restricted to the asynchronous fragment in which we only have $\multimap$-, Π-, $\top$-, and &-types. An exception is the Concurrent Logical Framework [8], which treats synchronous connectives resembling our I-, $\otimes$-, Σ-, and !-types. An account of additive disjunctions and identity types is missing entirely.

On the other hand, similar ideas, this time at the level of categorical semantics and specific models (from homotopy theory, algebra, and physics), have emerged in the mathematical community [9,10,11,12]. In these models, as with Girard, a notion of comprehension was missing and, with that, a notion of identity type. Although, in the past year, some suggestions have been made on the nLab and nForum of possible connections between the syntactic and semantic work, no account of the correspondence was published, as far as the author is aware.

The point of this paper[1] is to close this gap between syntax and semantics and to pave the way for a proper semantic analysis of linear type dependency, treating a range of type formers including the crucial Id-types[2]. Firstly, in section 2, we present a syntax, intuitionistic linear dependent type theory (ILDTT), a natural blend of the dual intuitionistic linear logic (DILL) [15] and dependent type theory (DTT) [16] which generalises both. Secondly, in section 3, we present a complete categorical semantics, an obvious combination of linear/non-linear adjunctions [15] and comprehension categories [17]. Finally, in section 4, an important class of models is studied: families with values in a symmetric monoidal category.

[1] This paper is based on the technical report [13] where proofs and more discussion can be found. Independently, Krishnaswami et al. [14] developed a roughly equivalent syntax and gave an operational rather than a denotational semantics. There, type dependency is added to Benton's LNL calculus, rather than to DILL.

[2] To be precise: extensional Id-types. Intensional Id-types remain a topic of investigation, due to the subtlety of dependent elimination rules in a linear setting.

2 Syntax

We assume the reader has some familiarity with the formal syntax of dependent type theory and linear type theory. In particular, we will not go into syntactic details like α-conversion, name binding, capture-free substitution of a for x in t (write $t[a/x]$), and pre-syntax. Details on all of these topics can be found in [16].

We next present the formal syntax of ILDTT. We start with a presentation of the judgements that will represent the propositions in the language and then discuss its rules of inference: first its structural core, then the logical rules for a series of optional type formers. We conclude this section with a few basic results about the syntax.

Judgements We adopt a notation $\Delta; \Xi$ for contexts, where Δ is 'an intuitionistic region' and Ξ is 'a linear region', as in DILL [15]. The idea will be that we have an empty context and can extend an existing context $\Delta; \Xi$ with both intuitionistic and linear types that are allowed to depend on Δ.

Our language will express judgements of the following six forms.

ILDTT judgement	**Intended meaning**
$\vdash \Delta; \Xi$ ctxt	$\Delta; \Xi$ is a valid context
$\Delta; \cdot \vdash A$ type	A is a type in (intuitionistic) context Δ
$\Delta; \Xi \vdash a : A$	a is a term of type A in context $\Delta; \Xi$
$\vdash \Delta; \Xi \equiv \Delta'; \Xi'$ ctxt	$\Delta; \Xi$ and $\Delta'; \Xi'$ are judgementally equal contexts
$\Delta; \cdot \vdash A \equiv A'$ type	A and A' are judgementally equal types in (intuitionistic) context Δ
$\Delta; \Xi \vdash a \equiv a' : A$	a and a' are judgementally equal terms of type A in context $\Delta; \Xi$

Fig. 1. Judgements of ILDTT.

Structural Rules We will use the following structural rules, which are essentially the structural rules of dependent type theory where some rules appear in both an intuitionistic and a linear form. We present the rules per group, with their names, from left-to-right, top-to-bottom.

Rules for context formation (C-Emp, Int-C-Ext, Int-C-Ext-Eq, Lin-C-Ext, Lin-C-Ext-Eq):

$$\frac{}{\cdot; \cdot \text{ ctxt}}$$

$$\frac{\vdash \Delta; \Xi \text{ ctxt} \qquad \Delta; \cdot \vdash A \text{ type}}{\vdash \Delta, x : A; \Xi \text{ ctxt}} \qquad \frac{\Delta; \Xi \equiv \Delta'; \Xi' \text{ ctxt} \qquad \Delta; \cdot \vdash A \equiv B \text{ type}}{\vdash \Delta, x : A; \Xi \equiv \Delta', y : B; \Xi' \text{ ctxt}}$$

$$\frac{\vdash \Delta; \Xi \text{ ctxt} \qquad \Delta; \cdot \vdash A \text{ type}}{\vdash \Delta; \Xi, x : A \text{ ctxt}} \qquad \frac{\Delta; \Xi \equiv \Delta'; \Xi' \text{ ctxt} \qquad \Delta; \cdot \vdash A \equiv B \text{ type}}{\vdash \Delta; \Xi, x : A \equiv \Delta'; \Xi', y : B \text{ ctxt}}$$

Variable declaration/axiom rules (Int-Var, Lin-Var):

$$\frac{\Delta, x : A, \Delta'; \cdot \text{ ctxt}}{\Delta, x : A, \Delta'; \cdot \vdash x : A} \qquad \frac{\Delta; x : A \text{ ctxt}}{\Delta; x : A \vdash x : A}$$

Fig. 2. Context formation and variable declaration rules.

The standard rules expressing that judgemental equality is an equivalence relation (C-Eq-R, C-Eq-S, C-Eq-T, Ty-Eq-R, Ty-Eq-S, Ty-Eq-T, Tm-Eq-R, Tm-Eq-S, Tm-Eq-T):

$$\frac{\vdash \Delta; \Xi \text{ ctxt}}{\vdash \Delta; \Xi \equiv \Delta; \Xi \text{ ctxt}} \qquad \frac{\vdash \Delta; \Xi \equiv \Delta'; \Xi' \text{ ctxt}}{\vdash \Delta'; \Xi' \equiv \Delta; \Xi \text{ ctxt}}$$

$$\frac{\vdash \Delta; \Xi \equiv \Delta'; \Xi' \text{ ctxt} \qquad \vdash \Delta'; \Xi' \equiv \Delta''; \Xi'' \text{ ctxt}}{\vdash \Delta; \Xi \equiv \Delta''; \Xi'' \text{ ctxt}}$$

$$\frac{\Delta; \Xi \vdash A \text{ type}}{\Delta; \Xi \vdash A \equiv A \text{ type}} \qquad \frac{\Delta; \Xi \vdash A \equiv A' \text{ type}}{\Delta; \Xi \vdash A' \equiv A \text{ type}}$$

$$\frac{\Delta; \Xi \vdash A \equiv A' \text{ type} \qquad \Delta; \Xi \vdash A' \equiv A'' \text{ type}}{\Delta; \Xi \vdash A \equiv A'' \text{ type}}$$

$$\frac{\Delta; \Xi \vdash a : A}{\Delta; \Xi \vdash a \equiv a : A} \qquad \frac{\Delta; \Xi \vdash a \equiv a' : A}{\Delta; \Xi \vdash a' \equiv a : A}$$

$$\frac{\Delta; \Xi \vdash a \equiv a' : A \qquad \Delta; \Xi \vdash a' \equiv a'' : A}{\Delta; \Xi \vdash a \equiv a'' : A}$$

The standard rules relating typing and judgemental equality (Tm-Conv, Ty-Conv):

$$\frac{\Delta; \Xi \vdash a : A \qquad \vdash \Delta; \Xi \equiv \Delta; \Xi' \text{ ctxt} \qquad \Delta; \cdot \vdash A \equiv A' \text{ Type}}{\Delta'; \Xi' \vdash a : A'}$$

$$\frac{\Delta'; \cdot \vdash A \text{ type} \qquad \vdash \Delta; \cdot \equiv \Delta'; \cdot \text{ ctxt}}{\Delta'; \cdot \vdash A \text{ type}}$$

Fig. 3. A few standard rules for judgemental equality.

Exchange, weakening, and substitution rules (Int-Weak, Int-Exch, Lin-Exch, Int-Ty-Subst, Int-Ty-Subst-Eq, Int-Tm-Subst, Int-Tm-Subst-Eq, Lin-Tm-Subst, Lin-Tm-Subst-Eq):

$$\frac{\Delta, \Delta'; \Xi \vdash \mathcal{J} \qquad \Delta; \cdot \vdash A \text{ type}}{\Delta, x : A, \Delta'; \Xi \vdash \mathcal{J}}$$

$$\frac{\Delta, x : A, x' : A', \Delta'; \Xi \vdash \mathcal{J}}{\Delta, x' : A', x : A, \Delta'; \Xi \vdash \mathcal{J}} \qquad \frac{\Delta; \Xi, x : A, x' : A', \Xi' \vdash \mathcal{J}}{\Delta; \Xi, x' : A', x : A, \Xi' \vdash \mathcal{J}}$$

(if x is not free in A')

$$\frac{\Delta, x : A, \Delta'; \cdot \vdash B \text{ type} \qquad \Delta; \cdot \vdash a : A}{\Delta, \Delta'[a/x]; \cdot \vdash B[a/x] \text{ type}} \qquad \frac{\Delta, x : A, \Delta'; \cdot \vdash B \equiv B' \text{ type} \qquad \Delta; \cdot \vdash a : A}{\Delta, \Delta'[a/x]; \cdot \vdash B[a/x] \equiv B'[a/x] \text{ type}}$$

$$\frac{\Delta, x : A, \Delta'; \Xi \vdash b : B \qquad \Delta; \cdot \vdash a : A}{\Delta, \Delta'[a/x]; \Xi[a/x] \vdash b[a/x] : B[a/x]} \qquad \frac{\Delta, x : A, \Delta'; \Xi \vdash b \equiv b' : B \qquad \Delta; \cdot \vdash a : A}{\Delta, \Delta'[a/x]; \Xi \vdash b[a/x] \equiv b'[a/x] : B[a/x]}$$

$$\frac{\Delta; \Xi, x : A \vdash b : B \qquad \Delta; \Xi' \vdash a : A}{\Delta; \Xi, \Xi' \vdash b[a/x] : B} \qquad \frac{\Delta; \Xi, x : A \vdash b \equiv b' : B \qquad \Delta; \Xi' \vdash a : A}{\Delta; \Xi, \Xi' \vdash b[a/x] \equiv b'[a/x] : B}$$

Fig. 4. Exchange, weakening, and substitution rules. Here, $\mathcal{J}$ represents a statement of the form B type, $B \equiv B'$, $b : B$, or $b \equiv b' : B$, such that all judgements are well-formed.

Logical Rules We describe some (optional) type and term formers, for which we give type formation (denoted -F), introduction (-I), elimination (-E), computation rules (-C), and (judgemental) uniqueness principles (-U). We also assume the obvious rules to hold that state that the type formers and term formers respect judgemental equality. Moreover, $\Sigma_{!x:!A}$, $\Pi_{!x:!A}$, $\lambda_{!x:!A}$, and $\lambda_{x:A}$ are name binding operators, binding free occurences of x within their scope.

We demand -U-rules for the various type formers in this paper, as this allows us to give a natural categorical semantics. This includes Id-types: we study extensional identity types. In practice, when building a computational implementation of a type theory like ours, one would probably drop some of these rules to make the system decidable, which would correspond to switching to weak equivalents of the categorical constructions presented here.[3]

$$\frac{\Delta, x : A; \cdot \vdash B \text{ type}}{\Delta; \cdot \vdash \Sigma_{!x:!A} B \text{ type}} \qquad \frac{\Delta; \cdot \vdash C \text{ type} \quad \Delta; \Xi \vdash t : \Sigma_{!x:!A} B \quad \Delta, x : A; \Xi', y : B \vdash c : C}{\Delta; \Xi, \Xi' \vdash \text{let } t \text{ be } !x \otimes y \text{ in } c : C}$$

$$\frac{\Delta; \cdot \vdash a : A \qquad \Delta; \Xi \vdash b : B[a/x]}{\Delta; \Xi \vdash !a \otimes b : \Sigma_{!x:!A} B}$$

$$\frac{\Delta; \Xi \vdash \text{let } !a \otimes b \text{ be } !x \otimes y \text{ in } c : C}{\Delta; \Xi \vdash \text{let } !a \otimes b \text{ be } !x \otimes y \text{ in } c \equiv c[a/x, b/y] : C} \qquad \frac{\Delta; \Xi \vdash \text{let } t \text{ be } !x \otimes y \text{ in } !x \otimes y : \Sigma_{!x:!A} B}{\Delta; \Xi \vdash \text{let } t \text{ be } !x \otimes y \text{ in } !x \otimes y \equiv t : \Sigma_{!x:!A} B}$$

$$\frac{\Delta, x : A; \cdot \vdash B \text{ type}}{\Delta; \cdot \vdash \Pi_{!x:!A} B \text{ type}}$$

$$\frac{\vdash \Delta; \Xi \text{ ctxt} \qquad \Delta, x : A; \Xi \vdash b : B}{\Delta; \Xi \vdash \lambda_{!x:!A} b : \Pi_{!x:!A} B} \qquad \frac{\Delta; \cdot \vdash a : A \qquad \Delta; \Xi \vdash f : \Pi_{!x:!A} B}{\Delta; \Xi \vdash f(!a) : B[a/x]}$$

$$\frac{\Delta; \Xi \vdash (\lambda_{!x:!A} b)(!a) : B}{\Delta; \Xi \vdash (\lambda_{!x:!A} b)(!a) \equiv b[a/x] : B[a/x]} \qquad \frac{\Delta; \Xi \vdash \lambda_{!x:!A} f(!x) : \Pi_{!x:!A} B}{\Delta; \Xi \vdash f \equiv \lambda_{!x:!A} f(!x) : \Pi_{!x:!A} B}$$

$$\frac{\Delta; \cdot \vdash a : A \qquad \Delta; \cdot \vdash a' : A}{\Delta; \cdot \vdash \text{Id}_{!A}(a, a') \text{ type}} \qquad \frac{\Delta, x : A, x' : A; \cdot \vdash D \text{ type} \quad \Delta, z : A; \Xi \vdash d : D[z/x, z/x'] \quad \Delta; \cdot \vdash a : A \quad \Delta; \cdot \vdash a' : A \quad \Delta; \Xi' \vdash p : \text{Id}_{!A}(a, a')}{\Delta; \Xi[a/z], \Xi' \vdash \text{let } (a, a', p) \text{ be } (z, z, \text{refl}_{!z}) \text{ in } d : D[a/x, a'/x']}$$

$$\frac{\Delta; \cdot \vdash a : A}{\Delta; \cdot \vdash \text{refl}_{!a} : \text{Id}_{!A}(a, a)}$$

$$\frac{\Delta; \Xi \vdash \text{let } (a, a, \text{refl}_{!a}) \text{ be } (z, z, \text{refl}_{!z}) \text{ in } d : D[a/x, a/x']}{\Delta; \Xi \vdash \text{let } (a, a, \text{refl}_{!a}) \text{ be } (z, z, \text{refl}_{!z}) \text{ in } d \equiv d[a/z] : D[a/x, a/x']}$$

$$\frac{\Delta, x : A, x' : A; \Xi, z : \text{Id}_{!A}(x, x') \vdash \text{let } (x, x', z) \text{ be } (x, x, \text{refl}_{!x}) \text{ in } c[x/x', \text{refl}_{!x}/z] : C}{\Delta, x : A, x' : A; \Xi, z : \text{Id}_{!A}(x, x') \vdash \text{let } (x, x', z) \text{ be } (x, x, \text{refl}_{!x}) \text{ in } c[x/x', \text{refl}_{!x}/z] \equiv c : C}$$

Fig. 5. Rules for linear equivalents of some of the usual type formers from DTT (Σ-F, -I, -E, -C, -U, Π-F, -I, -E, -C, -U, Id-F, -I, -E, -C, -U).

[3] In that case, in DTT, one would usually demand some stronger 'dependent' elimination rules, which would make propositional equivalents of the -U-rules provable, adding some extensionality to the system, while preserving its computational properties. Such rules are problematic in ILDTT, however, both from a syntactic and semantic point of view and a further investigation is warranted here.

$$\frac{}{\Delta;\cdot\vdash I\ \mathrm{type}}\qquad \frac{\Delta;\cdot\vdash I\ \mathrm{type}}{\Delta;\cdot\vdash *:I}\qquad \frac{\Delta;\Xi'\vdash t:I\qquad \Delta;\Xi\vdash a:A}{\Delta;\Xi,\Xi'\vdash \mathrm{let}\ t\ \mathrm{be}\ *\ \mathrm{in}\ a:A}$$

$$\frac{\Delta;\Xi\vdash \mathrm{let}\ *\ \mathrm{be}\ *\ \mathrm{in}\ a:A}{\Delta;\Xi\vdash \mathrm{let}\ *\ \mathrm{be}\ *\ \mathrm{in}\ a\equiv a:A}\qquad \frac{\Delta;\Xi\vdash \mathrm{let}\ t\ \mathrm{be}\ *\ \mathrm{in}\ *:I}{\Delta;\Xi\vdash \mathrm{let}\ t\ \mathrm{be}\ *\ \mathrm{in}\ *\equiv t:I}$$

$$\frac{\Delta;\cdot\vdash A\ \mathrm{type}\qquad \Delta;\cdot\vdash B\ \mathrm{type}}{\Delta;\cdot\vdash A\otimes B\ \mathrm{type}}$$

$$\frac{\Delta;\Xi\vdash a:A\qquad \Delta;\Xi'\vdash b:B}{\Delta;\Xi,\Xi'\vdash a\otimes b:A\otimes B}\qquad \frac{\Delta;\Xi\vdash t:A\otimes B\qquad \Delta;\Xi',x:A,y:B\vdash c:C}{\Delta;\Xi,\Xi'\vdash \mathrm{let}\ t\ \mathrm{be}\ x\otimes y\ \mathrm{in}\ c:C}$$

$$\frac{\Delta;\Xi\vdash \mathrm{let}\ a\otimes b\ \mathrm{be}\ x\otimes y\ \mathrm{in}\ c:C}{\Delta;\Xi\vdash \mathrm{let}\ a\otimes b\ \mathrm{be}\ x\otimes y\ \mathrm{in}\ c\equiv c[a/x,b/y]:C}\qquad \frac{\Delta;\Xi\vdash \mathrm{let}\ t\ \mathrm{be}\ x\otimes y\ \mathrm{in}\ x\otimes y:A\otimes B}{\Delta;\Xi\vdash \mathrm{let}\ t\ \mathrm{be}\ x\otimes y\ \mathrm{in}\ x\otimes y\equiv t:A\otimes B}$$

$$\frac{\Delta;\cdot\vdash A\ \mathrm{type}\qquad \Delta;\cdot\vdash B\ \mathrm{type}}{\Delta;\cdot\vdash A\multimap B\ \mathrm{type}}$$

$$\frac{\Delta;\Xi,x:A\vdash b:B}{\Delta;\Xi\vdash \lambda_{x:A}b:A\multimap B}\qquad \frac{\Delta;\Xi\vdash f:A\multimap B\qquad \Delta;\Xi'\vdash a:A}{\Delta;\Xi,\Xi'\vdash f(a):B}$$

$$\frac{\Delta;\Xi\vdash (\lambda_{x:A}b)(a):B}{\Delta;\Xi\vdash (\lambda_{x:A}b)(a)\equiv b[a/x]:B}\qquad \frac{\Delta;\Xi\vdash \lambda_{x:A}fx:A\multimap B}{\Delta;\Xi\vdash \lambda_{x:A}fx\equiv f:A\multimap B}$$

$$\frac{}{\Delta;\cdot\vdash \top\ \mathrm{type}}\qquad \frac{\Delta;\Xi\ \mathrm{ctxt}}{\Delta;\Xi\vdash\langle\rangle:\top}\qquad \frac{\Delta;\Xi\vdash t:\top}{\Delta;\Xi\vdash t\equiv\langle\rangle:\top}$$

$$\frac{\Delta;\cdot\vdash A\ \mathrm{type}\qquad \Delta;\cdot\vdash B\ \mathrm{type}}{\Delta;\cdot\vdash A\&B\ \mathrm{type}}\qquad \frac{\Delta;\Xi\vdash a:A\qquad \Delta;\Xi\vdash b:B}{\Delta;\Xi\vdash\langle a,b\rangle:A\&B}$$

$$\frac{\Delta;\Xi\vdash t:A\&B}{\Delta;\Xi\vdash \mathrm{fst}(t):A}\qquad \frac{\Delta;\Xi\vdash t:A\&B}{\Delta;\Xi\vdash \mathrm{snd}(t):B}$$

$$\frac{\Delta;\Xi\vdash \mathrm{fst}(\langle a,b\rangle):A}{\Delta;\Xi\vdash \mathrm{fst}(\langle a,b\rangle)\equiv a:A}\qquad \frac{\Delta;\Xi\vdash \mathrm{snd}(\langle a,b\rangle):B}{\Delta;\Xi\vdash \mathrm{snd}(\langle a,b\rangle)\equiv b:B}$$

$$\frac{\Delta;\Xi\vdash\langle \mathrm{fst}(t),\mathrm{snd}(t)\rangle:A\&B}{\Delta;\Xi\vdash\langle \mathrm{fst}(t),\mathrm{snd}(t)\rangle\equiv t:A\&B}$$

$$\frac{}{\Delta;\cdot\vdash 0\ \mathrm{type}}\qquad \frac{\Delta;\Xi\vdash t:0}{\Delta;\Xi,\Xi'\vdash \mathrm{false}(t):B}\qquad \frac{\Delta;\Xi\vdash t:0}{\Delta;\Xi\vdash \mathrm{false}(t)\equiv t:0}$$

$$\frac{\Delta;\cdot\vdash A\ \mathrm{type}\qquad \Delta;\cdot\vdash B\ \mathrm{type}}{\Delta;\cdot\vdash A\oplus B\ \mathrm{type}}$$

$$\frac{\Delta;\Xi\vdash a:A}{\Delta;\Xi\vdash \mathrm{inl}(a):A\oplus B}\qquad \frac{\Delta;\Xi\vdash b:B}{\Delta;\Xi\vdash \mathrm{inr}(b):A\oplus B}$$

$$\frac{\Delta;\Xi,x:A\vdash c:C\qquad \Delta;\Xi,y:B\vdash d:C\qquad \Delta;\Xi'\vdash t:A\oplus B}{\Delta;\Xi,\Xi'\vdash \mathrm{case}\ t\ \mathrm{of}\ \mathrm{inl}(x)\to c\ ||\ \mathrm{inr}(y)\to d:C}$$

$$\frac{\Delta;\Xi,\Xi'\vdash \mathrm{case}\ \mathrm{inl}(a)\ \mathrm{of}\ \mathrm{inl}(x)\to c\ ||\ \mathrm{inr}(y)\to d:C}{\Delta;\Xi,\Xi'\vdash \mathrm{case}\ \mathrm{inl}(a)\ \mathrm{of}\ \mathrm{inl}(x)\to c\ ||\ \mathrm{inr}(y)\to d\equiv c[a/x]:C}$$

$$\frac{\Delta;\Xi,\Xi'\vdash \mathrm{case}\ \mathrm{inr}(b)\ \mathrm{of}\ \mathrm{inl}(x)\to c\ ||\ \mathrm{inr}(y)\to d:C}{\Delta;\Xi,\Xi'\vdash \mathrm{case}\ \mathrm{inr}(b)\ \mathrm{of}\ \mathrm{inl}(x)\to c\ ||\ \mathrm{inr}(y)\to d\equiv d[b/y]:C}$$

$$\frac{\Delta;\Xi,\Xi'\vdash \mathrm{case}\ t\ \mathrm{of}\ \mathrm{inl}(x)\to \mathrm{inl}(x)\ ||\ \mathrm{inr}(y)\to \mathrm{inr}(y):A\oplus B}{\Delta;\Xi,\Xi'\vdash \mathrm{case}\ t\ \mathrm{of}\ \mathrm{inl}(x)\to \mathrm{inl}(x)\ ||\ \mathrm{inr}(y)\to \mathrm{inr}(y)\equiv t:A\oplus B}$$

$$\frac{\Delta;\cdot \vdash A \text{ type}}{\Delta;\cdot \vdash !A \text{ type}} \qquad \frac{\Delta;\cdot \vdash a : A}{\Delta;\cdot \vdash !a : !A} \qquad \frac{\Delta;\Xi \vdash t : !A \qquad \Delta, x : A;\Xi' \vdash b : B}{\Delta;\Xi,\Xi' \vdash \text{let } t \text{ be } !x \text{ in } b : B}$$

$$\frac{\Delta;\Xi \vdash \text{let } !a \text{ be } !x \text{ in } b : B}{\Delta;\Xi \vdash \text{let } !a \text{ be } !x \text{ in } b \equiv b[a/x] : B} \qquad \frac{\Delta;\Xi \vdash \text{let } t \text{ be } !x \text{ in } !x : !A}{\Delta;\Xi \vdash \text{let } t \text{ be } !x \text{ in } !x \equiv t : !A}$$

Fig. 6. Rules for the usual linear type formers in each context (I-F, -I, -E, -C, -U, $\otimes$-F, -I, -E, -C, -U, $\multimap$-F, -I, -E, -C, -U, $\top$-F, -I, -U, &-F, -I, -E1, -E2, -C1, -C2, -U, 0-F, -E, -U, $\oplus$-F, -I1, -I2, -E, -C1, -C2, -U, !-F, -I, -E, -C, -U).

Finally, we add rules that say we have all the possible commuting conversions, which from a syntactic point of view restore the subformula property and from a semantic point of view say that our rules are natural transformations (between hom-functors), which simplifies the categorical semantics significantly. We represent these schematically, following [15]. That is, if $C[-]$ is a linear program context, i.e. a context built without using !, then (abusing notation and dealing with all the let be in -constructors in one go) the following rules hold.

$$\frac{\Delta;\Xi \vdash C[\text{let } a \text{ be } b \text{ in } c] : D}{\Delta;\Xi \vdash C[\text{let } a \text{ be } b \text{ in } c] \equiv \text{let } a \text{ be } b \text{ in } C[c] : D} \quad \text{if } C[-] \text{ does not bind any free variables in } a \text{ or } b;$$

$$\frac{\Delta;\Xi \vdash C[\text{false}(t)] : D}{\Delta;\Xi \vdash C[\text{false}(t)] \equiv \text{false}(t) : D} \quad \text{if } C[-] \text{ does not bind any free variables in } t;$$

$$\frac{\Delta;\Xi \vdash C[\text{case } t \text{ of inl}(x) \to c \,||\, \text{inr}(y) \to d] : D}{\Delta;\Xi \vdash C[\text{case } t \text{ of inl}(x) \to c \,||\, \text{inr}(y) \to d] \equiv \text{case } t \text{ of inl}(x) \to C[c] \,||\, \text{inr}(y) \to C[d] : D}$$

if $C[-]$ does not bind any free variables in t or x or y.

Fig. 7. Commuting conversions.

Remark 1. Note that all type formers that are defined context-wise (I, $\otimes$, $\multimap$, $\top$, &, 0, $\oplus$, and !) are automatically preserved under the substitutions from Int-Ty-Subst (up to canonical isomorphism[4]), in the sense that $F(A_1,\ldots,A_n)[a/x]$ is isomorphic to $F(A_1[a/x],\ldots,A_n[a/x])$ for an n-ary type former F. Similarly, for $T = \Sigma$ or Π, we have that $(T_{!y:!B}C)[a/x]$ is isomorphic to $T_{!y:!B[a/x]}C[a/x]$ and $(Id_{!B}(b,b'))[a/x]$ is isomorphic to $Id_{!B[a/x]}(b[a/x],b'[a/x])$. This gives us Beck-Chevalley conditions in the categorical semantics.

Remark 2. The reader can note that the usual formulation of universes for DTT transfers very naturally to ILDTT, giving us a notion of universes for linear types. This allows us to write rules for forming types as rules for forming terms, as usual. We do not choose this approach and define the various type formers in the setting without universes.

[4] By an isomorphism of types $\Delta;\cdot \vdash A$ type and $\Delta;\cdot \vdash B$ type in context Δ, we here mean a pair of terms $\Delta; x : A \vdash f : B$ and $\Delta; y : B \vdash g : A$ together with a pair of judgemental equalities $\Delta; x : A \vdash g[f/y] \equiv x : A$ and $\Delta; y : B \vdash f[g/x] \equiv y : B$.

Some Basic Results As the focus of this paper is the syntax-semantics correspondence, we will only briefly state a few syntactic results. For some standard metatheoretic properties for (a system equivalent to) the $\multimap, \Pi, \top, \&$-fragment of our syntax, we refer the reader to [4]. Standard techniques and some small adaptations of the system should be enough to extend the results to all of ILDTT.

We will only note the consistency of ILDTT both as a type theory (not, for all $\Delta; \Xi \vdash a, a' : A$, $\Delta; \Xi \vdash a \equiv a' : A$) and as a logic (ILDTT does not prove that every type is inhabited).

Theorem 1 (Consistency). *ILDTT with all its type formers is consistent, both as a type theory and as a logic.*

Proof (sketch). This follows from model-theoretic considerations. Later, in section 3, we shall see that our model theory encompasses that of DTT, for which we have models exhibiting both types of consistency.

To give the reader some intuition for these linear Π- and Σ-types, we suggest the following two interpretations.

Theorem 2 (Π and Σ as Dependent $!(-) \multimap (-)$ and $!(-) \otimes (-)$). *Suppose we have $!$-types. Let $\Delta, x : A; \cdot \vdash B$ type, where x is not free in B. Then,*

1. *$\Pi_{!x:!A} B$ is isomorphic to $!A \multimap B$, if we have Π-types and $\multimap$-types;*
2. *$\Sigma_{!x:!A} B$ is isomorphic to $!A \otimes B$, if we have Σ-types and $\otimes$-types.*

In particular, we have the following stronger version of a special case.

Theorem 3 ($!$ as ΣI). *Suppose we have Σ- and I-types. Let $\Delta; \cdot \vdash A$ type. Then, $\Sigma_{!x:!A} I$ satisfies the rules for $!A$. Conversely, if we have $!$- and I-types, then $!A$ satisfies the rules for $\Sigma_{!x:!A} I$.*

A second interpretation is that Π and Σ generalise $\&$ and $\oplus$. Indeed, the idea is that that (or their infinitary equivalents) is what they reduce to when taken over discrete types. The subtlety in this result is the definition of a discrete type. The same phenomenon is observed in a different context in section 4.

For our purposes, a discrete type is a strong sum of $\top$ (a sum with a dependent -E-rule). Let us for simplicity limit ourselves to the binary case. For us, the discrete type with two elements will be $2 = \top \oplus \top$, where $\oplus$ has a strong/dependent -E-rule (note that this is not our $\oplus$-E). Explicitly, 2 is a type with the following -F-, -I-, and -E-rules (and the obvious -C- and -U-rules):

$$\frac{}{\Delta; \cdot \vdash 2 \text{ type}} \qquad \frac{}{\Delta; \cdot \vdash \text{tt} : 2} \qquad \frac{}{\Delta; \cdot \vdash \text{ff} : 2}$$

$$\frac{\Delta, x : 2; \cdot \vdash A \text{ type} \qquad \Delta; \cdot \vdash t : 2 \qquad \Delta; \Xi \vdash a_{\text{tt}} : A[\text{tt}/x] \qquad \Delta; \Xi \vdash a_{\text{ff}} : A[\text{ff}/x]}{\Delta; \Xi \vdash \text{if } t \text{ then } a_{\text{tt}} \text{ else } a_{\text{ff}} : A[t/x]}$$

Fig. 8. Rules for a discrete type 2, with -C- and -U-rules omitted for reasons of space.

Theorem 4 (Π and Σ as Infinitary Non-Discrete $\&$ and $\oplus$). *If we have a discrete type 2 and a type family $\Delta, x : 2; \cdot \vdash A$, then*

1. *$\Pi_{!x:!2} A$ satisfies the rules for $A[\text{tt}/x] \& A[\text{ff}/x]$;*
2. *$\Sigma_{!x:!2} A$ satisfies the rules for $A[\text{tt}/x] \oplus A[\text{ff}/x]$.*

3 Categorical Semantics

We now introduce a notion of categorical model for which soundness and completeness results hold with respect to the syntax of ILDTT in presence of I- and $\otimes$-types[5]. This notion of model will prove to be particularly useful when thinking about various (extensional) type formers.

Definition 1. *By a* strict indexed symmetric monoidal category with comprehension*, we will mean the following data.*

1. *A category $\mathcal{C}$ with a terminal object $\cdot$.*
2. *A strict indexed symmetric monoidal category $\mathcal{L}$ over $\mathcal{C}$, i.e. a contravariant functor $\mathcal{L}$ into the category* SMCat *of (small) symmetric monoidal categories and strong monoidal functors $\mathcal{C}^{op} \xrightarrow{\mathcal{L}}$ SMCat. We will also write $-\{f\} := \mathcal{L}(f)$ for the action of $\mathcal{L}$ on a morphism f of $\mathcal{C}$.*
3. *A* comprehension schema*, i.e. for each $\Delta \in \mathrm{ob}(\mathcal{C})$ and $A \in \mathrm{ob}(\mathcal{L}(\Delta))$ a representation for the functor*

$$x \mapsto \mathcal{L}(\mathrm{dom}(x))(I, A\{x\}) : (\mathcal{C}/\Delta)^{op} \longrightarrow \mathrm{Set}.$$

We will write its representing object[6] *$\Delta.A \xrightarrow{\mathbf{p}_{\Delta,A}} \Delta \in \mathrm{ob}(\mathcal{C}/\Delta)$ and universal element $\mathbf{v}_{\Delta,A} \in \mathcal{L}(\Delta.A)(I, A\{\mathbf{p}_{\Delta,A}\})$. We will write $a \mapsto \langle f, a\rangle$ for the isomorphism $\mathcal{L}(\Delta')(I, A\{f\}) \cong \mathcal{C}/\Delta(f, \mathbf{p}_{\Delta,A})$, if $\Delta' \xrightarrow{f} \Delta$.*

Remark 3. Note that this notion of model reduces to a standard notion of model for DTT in the case the monoidal structures on the fibre categories are Cartesian: a reformulation of split comprehension categories with 1- and $\times$-types. To get a precise fit with the syntax, the extra demand called "fullness" is usually put on these [17]. The fact that we leave out this last condition precisely allows for non-trivial !-types (i.e. ones such that $!A \ncong A$) in our models of ILDTT. Every model of DTT is, in particular, a (degenerate) model of ILDTT, though. We will see that the type formers of ILDTT also generalise those of DTT.

Theorem 5 (Soundness). *We can soundly interpret ILDTT with I- and $\otimes$-types in a strict indexed symmetric monoidal category $(\mathcal{C}, \mathcal{L})$ with comprehension.*

Proof (sketch). The idea is that a context $\Delta; \Xi$ will be (inductively) interpreted by a pair of objects $[\![\Delta]\!] \in \mathrm{ob}(\mathcal{C})$, $[\![\Xi]\!] \in \mathrm{ob}(\mathcal{L}([\![\Delta]\!])$, a type A in context $\Delta; \cdot$ by an object $[\![A]\!]$ of $\mathcal{L}([\![\Delta]\!])$, and a term $a : A$ in context $\Delta; \Xi$ by a morphism $[\![\Xi]\!] \xrightarrow{[\![a]\!]} [\![A]\!] \in \mathcal{L}[\![\Delta]\!])$. Generally, the interpretation of the propositional linear type theory in intuitionistic context $\Delta; \cdot$ will happen in $\mathcal{L}(\Delta)$ as would be expected.

The crux is that Int-C-Ext ($[\![\Delta, x : A]\!] := \mathrm{dom}(\mathbf{p}_{[\![\Delta]\!],[\![A]\!]})$), Int-Var ($[\![\Delta, x : A; \cdot \vdash x : A]\!] := \mathbf{v}_{\Delta,A}$), and Int-Subst (by $\mathcal{L}(\langle \mathrm{id}_\Delta, a\rangle)$ are interpreted through the comprehension, as is Int-Weak (through $\mathcal{L}$ of the obvious morphism in $\mathcal{C}$).

Finally, Soundness is a trivial verification.

[5] In case we are interested in the case without I- and $\otimes$-types, the semantics easily generalises to strict indexed symmetric multicategories with comprehension.

[6] Really, $\Delta.MA \xrightarrow{\mathbf{p}_{\Delta,MA}} \Delta$ would be a better notation, where we think of $L \dashv M$ as an adjunction inducing !, but it would be very verbose.

Theorem 6 (Completeness). *In fact, this interpretation is complete.*

Proof (sketch). We see this through the construction of a syntactic category.

In fact, we would like to say that the syntax is even an internal language for such categories. This is almost true, can be made entirely true by either putting the restriction on our notion of model that excludes any non-trivial morphisms into objects that are not of the form $\Delta.A$. Alternatively, we can extend the syntax to talk about context morphisms explicitly [18]. Following the DTT tradition, we have opted against the latter.

We will next characterise the categorical description of the various type formers. First, we note the following.

Theorem 7 (Comprehension Functor). *A comprehension schema* $(\mathbf{p},\mathbf{v})$ *on a strict indexed symmetric monoidal category* $(\mathcal{C},\mathcal{L})$ *defines a morphism* $\mathcal{L}\xrightarrow{M}\mathcal{I}$ *of indexed categories, where* $\mathcal{I}$ *is the full sub-indexed category of* $\mathcal{C}/-$ *(by making a choice of pullbacks) on the objects of the form* $\mathbf{p}_{\Delta,A}$ *and where* $M_\Delta(A\xrightarrow{a}B):=\mathbf{p}_{\Delta,A}\xrightarrow{\langle\mathbf{p}_{\Delta,A},a\{\mathbf{p}_{\Delta,A}\}\circ\mathbf{v}_{\Delta,A}\rangle}\mathbf{p}_{\Delta,B}$.

Note that $\mathcal{I}$ is a display map category and hence a model of DTT [17]. We will think of it as the intuitionistic content of $\mathcal{L}$. We will see that the comprehension functor will give us a unique candidate for !-types: $!:=LM$, where $L\dashv M$ is a monoidal adjunction. We conclude that, in ILDTT, the !-modality is uniquely determined by the indexing. This is worth noting, because, in propositional linear type theory, we might have many different candidates for !-types.

Theorem 8 (Semantic Type Formers). *For the other type formers, we have the following. A model* $(\mathcal{C},\mathcal{L},\mathbf{p},\mathbf{v})$ *of ILDTT with* I*- and* $\otimes$*-types...*

1. *...supports* Σ*-types iff all the pullback functors* $\mathcal{L}(\mathbf{p}_{\Delta,A})$ *have left adjoints* $\Sigma_{!A}$ *that satisfy the Beck-Chevalley condition in the sense that the canonical map* $\Sigma_{!A\{f\}}\circ\mathcal{L}(\mathbf{q}_{f,A})\longrightarrow\mathcal{L}(f)\circ\Sigma_{!A}$ *is an iso, where* $\Delta'\xrightarrow{f}\Delta$ *and* $\mathbf{q}_{f,A}:=\langle f\circ\mathbf{p}_{\Delta',A\{f\}},\mathbf{v}_{\Delta',A\{f\}}\rangle$*, and that satisfy Frobenius reciprocity in the sense that the canonical morphism* $\Sigma_{!A}(\Xi'\{\mathbf{p}_{\Delta,A}\}\otimes B)\longrightarrow\Xi'\otimes\Sigma_{!A}B$ *is an isomorphism , for all* $\Xi'\in\mathcal{L}(\Delta)$*,* $B\in\mathcal{L}(\Delta.A)$ *.*
2. *...supports* Π*-types iff all the pullback functors* $\mathcal{L}(\mathbf{p}_{\Delta,A})$ *have right adjoints* $\Pi_{!A}$ *that satisfy the dual Beck-Chevalley condition for pullbacks of the form* $(*)$*: the canonical* $\mathcal{L}(f)\circ\Pi_{!A}\longrightarrow\Pi_{!A\{f\}}\circ\mathcal{L}(\mathbf{q}_{f,A})$ *is an iso.*
3. *...supports* $\multimap$*-types iff* $\mathcal{L}$ *factors over the category* SMCCat *of symmetric monoidal closed categories and their homomorphisms.*
4. *...supports* $\top$*- and* $\&$*-types iff* $\mathcal{L}$ *factors over the category* SMCCat *of Cartesian categories with symmetric monoidal structure and their homomorphisms.*
5. *...supports* 0*- and* $\oplus$*-types iff* $\mathcal{L}$ *factors over the category* dSMcCCat *of co-Cartesian categories with a distributive symmetric monoidal structure and their homomorphisms.*

6. *...that supports $\multimap$-types, supports* !*-types iff all the comprehension functors* $\mathcal{L}(\Delta) \xrightarrow{M_\Delta} \mathcal{I}(\Delta)$ *have a strong monoidal left adjoint* $\mathcal{I}(\Delta) \xrightarrow{L_\Delta} \mathcal{L}(\Delta)$ *and* L_- *is a morphism of indexed categories: for all* $\Delta' \xrightarrow{f} \Delta \in \mathcal{C}$, $L_{\Delta'}\mathcal{I}(f) = \mathcal{L}(f)L_\Delta$. *Then* $!_\Delta := L_\Delta \circ M_\Delta$ *interprets the comodality* ! *in context* Δ.
7. *... that supports $\multimap$-types, supports Id-types iff for all* $A \in \mathrm{ob}(\mathcal{L}(\Delta))$, *we have left adjoints* $\mathrm{Id}_{!A} \dashv -\{\mathrm{diag}_{\Delta,A}\}$ *that satisfy a Beck-Chevalley condition:* $\mathrm{Id}_{!A\{f\}} \circ \mathcal{L}(\mathbf{q}_{f,A}) \longrightarrow \mathcal{L}(\mathbf{q}_{\mathbf{q}_{f,A},A\{\mathbf{p}_{\Delta,A}\}}) \circ \mathrm{Id}_{!A}$ *is an iso. Now,* $\mathrm{Id}_{!A}(I)$ *interprets* $\mathrm{Id}_{!A}(x,x')$. *Above,* $\Delta.A \xrightarrow{\mathrm{diag}_{\Delta,A} := \langle \mathrm{id}_{\Delta.A}, \mathbf{v}_{\Delta,A}\rangle} \Delta.A.A\{\mathbf{p}_{\Delta,A}\}$.

The semantics of ! suggests an alternative definition for the notion of a comprehension: if we have Σ-types in a strong sense, it is a derived notion!

Theorem 9 (Lawvere Comprehension). *Given a strict indexed monoidal category* $(\mathcal{C},\mathcal{L})$ *with left adjoints* Σ_{Lf} *to* $\mathcal{L}(f)$ *for arbitrary* $\Delta' \xrightarrow{f} \Delta \in \mathcal{C}$, *then we can define* $\mathcal{C}/\Delta \xrightarrow{L_\Delta} \mathcal{L}(\Delta)$ *by* $L_\Delta(-) := \Sigma_{L-}I$. *In that case,* $(\mathcal{C},\mathcal{L})$ *has a comprehension schema iff* L_Δ *has a right adjoint* M_Δ *(for which then* $M_{\Delta'} \circ \mathcal{L}(f) = \mathcal{L}(f) \circ M_\Delta$ *for all* $\Delta' \xrightarrow{f} \Delta \in \mathcal{C}$*). That is, our notion of comprehension generalises that of Lawvere [19]. Finally, if* Σ_{Lf} *satisfy Frobenius reciprocity and Beck-Chevalley, then* $(\mathcal{C},\mathcal{L})$ *supports comprehension iff it supports* !*-types.*

Proof (sketch). This follows trivially if we write out both the representability condition defining a comprehension and the adjointness condition for Σ_f.

We observe the following about the usual intuitionistic type formers in $\mathcal{I}$.

Theorem 10 (Type Formers in $\mathcal{I}$). $\mathcal{I}$ *supports* Σ*-types iff* $\mathrm{ob}(\mathcal{I}) \subset \mathrm{mor}(\mathcal{C})$ *is closed under binary compositions.* $\mathcal{I}$ *supports* Id*-types iff* $\mathrm{ob}(\mathcal{I})$ *is closed under post-composition with* $\mathrm{diag}_{\Delta,A}$. *If* $\mathcal{L}$ *supports* !*- and* Π*-types, then* $\mathcal{I}$ *supports* Π*-types. Moreover, type formers in* $\mathcal{I}$ *relate to those in* $\mathcal{L}$ *as follows, leaving out the subscripts of the indexed functors* $L \dashv M$*:*

$$\Sigma_{!A}!B \cong L(\Sigma_{MA}MB) \qquad \mathrm{Id}_{!A}(!B) \cong L\mathrm{Id}_{MA}(MB) \qquad M\Pi_{!B}C \cong \Pi_{MB}MC.$$

Remark 4 (Dependent Seely Isomorphisms?). It is easily seen that $M_\Delta(\top) = \mathrm{id}_\Delta$ and $M_\Delta(A\&B) = M_\Delta(A) \times M_\Delta(B)$, hence $!_\Delta\top = I$ and $!_\Delta(A\&B) = !_\Delta A \otimes !_\Delta B$.

Now, theorem 10 suggests similar Seely isomorphisms for Σ- and Id-types. Indeed, $\mathcal{I}$ supports Σ- respectively Id-types iff we have "additive" Σ- resp. Id-types, that is $\Sigma^{\&}_A B, \mathrm{Id}^{\&}_A(B) \in \mathrm{ob}(\mathcal{L})$ s.t.

$$M\Sigma^{\&}_A B \cong \Sigma_{MA}MB \quad \text{and hence} \quad !\Sigma^{\&}_A B \cong \Sigma^{\otimes}_{!A}!B \quad \text{resp.}$$

$$M\mathrm{Id}^{\&}_A(B) \cong \mathrm{Id}_{MA}(MB) \quad \text{and hence} \quad !\mathrm{Id}^{\&}_A(B) \cong \mathrm{Id}^{\otimes}_{!A}(!B),$$

where we write $\Sigma^\otimes$ and $\mathrm{Id}^\otimes$ for the usual multiplicative Σ- and Id-types[7].

We are in this situation and have to consider such additive Σ- and Id-types if $L_\cdot \dashv M_\cdot : \mathcal{L}(\cdot) \longrightarrow \mathcal{C}$ is the co-Kleisli adjunction of !. See [13] for more discussion.

[7] We call usual Id-types "multiplicative" connectives e.g. since $\mathrm{Id}^\otimes_{!A}(B) \cong \mathrm{Id}^\otimes_{!A}(I) \otimes B$. Similarly, if we have a suitable $\mathrm{Id}^{\&}_A(\top)$, we can define $\mathrm{Id}^{\&}_A(B) := \mathrm{Id}^{\&}_A(\top)\&B$.

4 Some Discrete Models: Monoidal Families

We discuss a simple class of models in terms of families with values in a symmetric monoidal category. On a logical level, what the construction boils down to is starting with a model $\mathcal{V}$ of a linear propositional logic and taking the cofree linear predicate logic on Set with values in this propositional logic. This important example illustrates how Σ- and Π-types can represent infinitary additive disjunctions and conjunctions. The model is discrete in nature, however, and, in that respect, is not representative for ILDTT.

Suppose $\mathcal{V}$ is a symmetric monoidal category. We can then consider a strict Set-indexed category, defined through the following enriched Yoneda embedding $\mathrm{Fam}(\mathcal{V}) := \mathcal{V}^- := \mathrm{SMCat}(-,\mathcal{V})$:

$$\mathrm{Set}^{op} \xrightarrow{\mathrm{Fam}(\mathcal{V})} \mathrm{SMCat} \qquad\qquad S \xrightarrow{f} S' \longmapsto \mathcal{V}^S \xleftarrow{-\circ f} \mathcal{V}^{S'}.$$

Note that this definition naturally extends to a functorial embedding Fam.

Theorem 11 (Families Model ILDTT). *The construction* Fam *adds type dependency on* Set *cofreely, in the sense that it is right adjoint to the forgetful functor* ev_1 *that evaluates a model of linear dependent type theory at the empty context to obtain a model of linear propositional type theory (where* $\mathrm{SMCat}^{\mathrm{Set}^{op}}_{\mathrm{compr}}$ *is the full subcategory of* $\mathrm{SMCat}^{\mathrm{Set}^{op}}$ *on the objects with comprehension):*

$$\mathrm{SMCat} \underset{\mathrm{Fam}}{\overset{\mathrm{ev}_1}{\underset{\subset}{\overset{\longleftarrow}{\bot}\!\!\longrightarrow}}} \mathrm{SMCat}^{\mathrm{Set}^{op}}_{\mathrm{compr}}.$$

Proof (sketch). The comprehension on $\mathrm{Fam}(\mathcal{V})$ is given by the obvious bijection

$$\mathrm{Fam}(\mathcal{V})(S)(I, B\{f\}) \cong \mathrm{prod}_{s\in S}\mathcal{V}(I, B(f(s))) \cong \mathrm{Set}/S'(f, \mathbf{p}_{S',B}),$$

where $\mathbf{p}_{S',B} := \mathrm{coprod}_{s'\in S'}\mathcal{V}(I, B(s')) \xrightarrow{\mathrm{fst}} S'$. The rest of the proof is a straightforward verification, where the adjunction relies on Set being well-pointed.

We express the existence of type formers in $\mathrm{Fam}(\mathcal{V})$ as conditions on $\mathcal{V}$. A characterisation of additive Σ- and Id-types can be found in [13].

Theorem 12 (Type Formers for Families). *$\mathcal{V}$ has small coproducts that distribute over $\otimes$ iff* $\mathrm{Fam}(\mathcal{V})$ *supports Σ-types. In that case,* $\mathrm{Fam}(\mathcal{V})$ *also supports* 0- *and $\oplus$-types (which correspond precisely to finite distributive coproducts).*

$\mathcal{V}$ has small products iff $\mathrm{Fam}(\mathcal{V})$ *supports Π-types. In that case,* $\mathrm{Fam}(\mathcal{V})$ *also supports $\top$- and $\&$-types (which correspond precisely to finite products).*

$\mathrm{Fam}(\mathcal{V})$ *supports $\multimap$-types iff $\mathcal{V}$ is monoidal closed.*

$\mathrm{Fam}(\mathcal{V})$ *supports* !*-types iff $\mathcal{V}$ has small coproducts of I that are preserved by $\otimes$ in the sense that the canonical morphism* $\mathrm{coprod}_S(\Xi' \otimes I) \longrightarrow \Xi' \otimes \mathrm{coprod}_S I$ *is an isomorphism for any $\Xi' \in \mathrm{ob}\,\mathcal{V}$ and $S \in \mathrm{ob}\,\mathrm{Set}$. In particular, if* $\mathrm{Fam}(\mathcal{V})$ *supports Σ-types, then it also supports* !*-types.*

$\mathrm{Fam}(\mathcal{V})$ *supports* Id*-types if $\mathcal{V}$ has an initial object. Supposing that $\mathcal{V}$ has a terminal object, the only if also holds.*

Proof (sketch). We supply some definitions and leave the rest to the reader.

$\top$-, &-, 0-, and $\oplus$-types are clear as (co)limits are pointwise in a functor category. $\multimap$-types are immediate as well from the previous section. We define $\Sigma_{Lf}(A)(s') := \mathrm{coprod}_{s \in f^{-1}(s')} A(s)$ and $\Pi_{Lf}(A)(s') = \mathrm{prod}_{s \in f^{-1}(s')} A(s)$. Then $\Sigma_{Lf} \dashv -\{f\} \dashv \Pi_{Lf}$. We define $\mathrm{Id}_{!A}(B)(s, a, a') := \begin{cases} B(s,a) \text{ if } a = a' \\ 0 \text{ else} \end{cases}$. Then, $\mathrm{Id}_{!A} \dashv -\{\mathrm{diag}_{S,A}\}$. Beck-Chevalley conditions are taken care of by the fact that subtitution is interpreted as precomposition. Finally, this leads to the definition $!A(s) := \mathrm{coprod}_{\mathcal{V}(I, A(s))} I$, which we can note only depends on $A(s)$.

Remark 5. Note that an obvious way to guarantee distributivity of coproducts over $\otimes$ is by demanding that $\mathcal{V}$ is monoidal closed.

Two simple concrete examples of $\mathcal{V}$ come to mind that accommodate all type formers and illustrate real *linear* type dependency: a category $\mathcal{V} = \mathrm{Vect}_F$ of vector spaces over a field F, with the tensor product, and the category $\mathcal{V} = \mathrm{Set}_*$ of pointed sets, with the smash product. All type formers get their obvious interpretation, but let us consider ! as it is a novelty of ILDTT that it gets uniquely determined by the indexing, while in propositional linear logic we might have several choices. In the first example, ! boils down to the following: $(!B)(s') = \mathrm{coprod}_{\mathrm{Vect}_F(F, B(s'))} F \cong \bigoplus_{B(s')} F$, i.e. taking the vector space freely spanned by all vectors. In the second example, $(!B)(s') = \mathrm{coprod}_{\mathrm{Set}_*(2_*, B(s'))} 2_* = \bigvee_{B(s')} 2_* = B(s') + \{*\}$, i.e. ! freely adds a new basepoint. These models show the following.

Theorem 13 (DTT,DILL$\subsetneq$ ILDTT). *ILDTT is a proper generalisation of DTT and DILL: we have inclusions of the classes of models DTT,DILL$\subsetneq$ILDTT.*

Although this class of models is important, it is clear that it only represents a limited part of the generality of ILDTT. Hence, we are in need of non-Cartsian models that are less discrete in nature, if we are hoping to observe interesting new phenomena arising from the connectives of linear dependent type theory. Some suggestions and work in progress will be discussed in the next section.

5 Conclusions and Future Work

We hope to have convinced the reader that linear dependent types fit very naturally in the landscape of existing type theories and that they admit a well-behaved semantic theory.

We have presented a system, ILDTT, that, on a syntactic level, is a natural blend between (intuitionistic) dependent type theory (DTT) and dual intuitionistic linear logic (DILL). On a semantic level, if one starts with the right notion of model for dependent types, the linear generalisation is obtained through the usual philosophy of passing from Cartesian to symmetric monoidal structures. The resulting notion of a model forms a natural blend between comprehension categories, modelling DTT, and linear-non-linear models of DILL.

It is very pleasing to see that all the syntactically natural rules for type formers are equivalent to their semantic counterparts that would be expected

based on the traditions of categorical logic of dependent types and linear types. In particular, from the point of view of logic, it is interesting to see that the categorical semantics seems to have a preference for multiplicative quantifiers.

Finally, we have shown that, as in the intuitionistic case, we can represent infinitary (additive) disjunctions and conjunctions in linear type theory, through cofree Σ- and Π-types, indexed over Set. In particular, this construction exhibits a family of non-trivial truly linear models of dependent types. Moreover, it shows that ILDTT properly extends both DILL and DTT.

Despite what might be expected from this paper, much of this work has been very semantically motivated, by specific models. In joint work with Samson Abramsky, a model of linear dependent types with comprehension has been constructed in a category of coherence spaces. Apart from the usual type constructors from linear logic, it also supports Σ-, Π-, and Id-types. A detailed account of this model will be made available soon.

In addition to providing a first non-trivial model of such a type system that goes properly beyond DILL and DTT and is semantically motivated, this work served as a stepping stone for a model in a category of games, which we developed together with Radha Jagadeesan and Samson Abramsky. This, in particular, provides a game semantics for dependent type theory.

An indexed category of spectra over topological spaces has been studied as a setting for stable homotopy theory [9,11]. It has been shown to admit I-, $\otimes$-, $\multimap$-, and Σ-types. The natural candidate for a comprehension adjunction, here, is that between the infinite suspension spectrum and the infinite loop space: $L \dashv M \quad = \quad \Sigma^\infty \dashv \Omega^\infty$. A detailed examination of the situation and an explanation of the relation with the Goodwillie calculus is desirable. This might fit in with our ultimate objective of a linear analysis of homotopy type theory.

Another fascinating possibility is that of models related to quantum mechanics. Non-dependent linear type theory has found interesting interpretations in quantum computation [20]. The question rises if the extension to dependent linear types has a natural counterpart in physics and could e.g. provide stronger type systems for quantum computing. Also suggestive is Schreiber's work [12], in which it is sketched how linear dependent types can serve as a language to talk about quantum field theory and quantisation in particular.

Finally, there are still plenty of theoretical questions within the type theory. Can we find interesting models with type dependency on the co-Kleisli category of ! and can we make sense of additive Σ- and Id-types, e.g. from the point of view of syntax? Or should we perhaps doubt the canonicity of the Girard translation and accept that dependent types are more naturally modeled in co-Eilenberg-Moore categories? Is there an equivalent of strong/dependent E-rules for ILDTT and how do we model interesting intensional Id-types? Does the Curry-Howard correspondence extend in its full glory: do we have a propositions-as-types interpretation of linear predicate logic in ILDTT? These questions need to be addressed by a combination of research into the formal system and study of specific models. We hope that the general framework we sketched will play its

part in connecting all the different sides of the story: from syntax to semantics; from computer science and logic to geometry and physics.

Acknowledgements

My thanks go out to Samson Abramsky and Radha Jagadeesan for the stimulating discussions and to Urs Schreiber for sparking my curiosity for this topic. I am indebted to the anonymous reviewers, whose comments have been very helpful. This research was supported by the EPSRC and the Clarendon Fund.

References

1. Martin-Löf, P.: An intuitionistic theory of types. Twenty-five years of constructive type theory **36** (1998) 127–172
2. Girard, J.Y.: Linear logic. Theoretical computer science **50**(1) (1987) 1–101
3. Program, T.U.F.: Homotopy Type Theory: Univalent Foundations of Mathematics. http://homotopytypetheory.org/book, Institute for Advanced Study (2013)
4. Cervesato, I., Pfenning, F.: A linear logical framework. In: LICS'96. Proceedings., IEEE (1996) 264–275
5. Dal Lago, U., Gaboardi, M.: Linear dependent types and relative completeness. In: LiCS 2011. Proceedings., IEEE (2011) 133–142
6. Petit, B., et al.: Linear dependent types in a call-by-value scenario. In: Proceedings of the 14th symposium on Principles and practice of declarative programming, ACM (2012) 115–126
7. Gaboardi, M., Haeberlen, A., Hsu, J., Narayan, A., Pierce, B.C.: Linear dependent types for differential privacy. In: ACM SIGPLAN Notices. Volume 48., ACM (2013) 357–370
8. Watkins, K., Cervesato, I., Pfenning, F., Walker, D.: A concurrent logical framework i: Judgments and properties. Technical report, DTIC Document (2003)
9. May, J.P., Sigurdsson, J.: Parametrized homotopy theory. Number 132. American Mathematical Soc. (2006)
10. Shulman, M.: Enriched indexed categories. Theory and Applications of Categories **28**(21) (2013) 616–695
11. Ponto, K., Shulman, M.: Duality and traces for indexed monoidal categories. Theory and Applications of Categories **26**(23) (2012) 582–659
12. Schreiber, U.: Quantization via linear homotopy types. arXiv preprint arXiv:1402.7041 (2014)
13. Vákár, M.: Syntax and semantics of linear dependent types. arXiv preprint arXiv:1405.0033 (2014) Original preprint from April 2014.
14. Krishnaswami, N.R., Pradic, P., Benton, N.: Integrating dependent and linear types. (July 2014) On: https://www.mpi-sws.org/ neelk/dlnl-paper.pdf.
15. Barber, A.: Dual intuitionistic linear logic. Technical Report ECS-LFCS-96-347, University of Edinburgh, Edinburgh (1996)
16. Hofmann, M.: Syntax and semantics of dependent types. In: Extensional Constructs in Intensional Type Theory. Springer (1997) 13–54
17. Jacobs, B.: Comprehension categories and the semantics of type dependency. Theoretical Computer Science **107**(2) (1993) 169–207
18. Pitts, A.M.: Categorical logic. In Abramsky, S., Gabbay, D.M., Maibaum, T.S.E., eds.: Handbook of Logic in Computer Science, Volume 5. Algebraic and Logical Structures. Oxford University Press (2000) 39–128

19. Lawvere, F.W.: Equality in hyperdoctrines and comprehension schema as an adjoint functor. Applications of Categorical Algebra **17** (1970) 1–14
20. Abramsky, S., Duncan, R.: A categorical quantum logic. Mathematical Structures in Computer Science **16**(3) (2006) 469–489 Preprint available at http://arxiv.org/abs/quant-ph/0512114.